RECEIVED
CENTRAL FAX CENTER

SEP 10 2008

In the claims:

For the Examiner's convenience, all pending claims are presented below with changes shown.

1-9. Cancelled

10. (Previously Presented) For a computer-executable program that operates on a data structure, where the data structure must have a required state at selected program points, a computer-implemented method of transforming said program comprising:

speculatively computing down-safety by ignoring branches in a control-flow graph if it is determined that the branches are fail edges, wherein the computation of down safety further comprises a lattice that distinguishes strict down-safety from speculative down-safety;

computing up-safety using the results of the down-safety calculation to determine where operations are speculatively available;

using the down-safety and up-safety to determine instructions to be inserted into the routine that set components in a stack, the stack to handle cleanup instructions for the routine; and

removing edges from one or more exceptional paths in the routine to eliminate the partial redundancy in the routine, the edges to be removed having purposes that are performed by the instructions.

11. (Cancelled)

12. (Previously Presented) The computer-implemented method of claim 10, further comprising determining whether the routine includes the cleanup instructions prior to computing the down-safety.

13. (Previously Presented) The computer-implemented method of claim 12, further comprising:

removing the edges from one or more exceptional paths if the routine does not include cleanup instructions; and

inserting ~~a prologue~~ one or more instructions at the beginning of the routine.

14. (Previously Presented) The computer-implemented method of claim 12, further comprising building a cleanup tree and cleanup states for the routine.

15. (Previously Presented) The computer-implemented method of claim 14, wherein the cleanup tree and cleanup states represents the stack at points where an exception may be thrown.

16. (Previously Presented) The computer-implemented method of claim 15, wherein the cleanup tree and cleanup states assist in the determining instructions to be inserted into the routine that set components in the stack.

17. (Previously Presented) The computer-implemented method of claim 10, wherein computing down-safety comprises:

computing a down-safety transfer function for each of the edges of the one or more exceptional path; and

solving flow equations for the down-safety transfer functions to produce a downsafe map from vertices in the routine to corresponding Boolean values.

18. (Previously Presented) The computer-implemented method of claim 17, wherein computing up-safety comprises:

computing an up-safety transfer function for each of the edges of the one or more exceptional paths; and

solving flow equations for the up-safety transfer functions to produce an upsafe map from vertices in the routine to corresponding Boolean values.

19. (Previously Presented) The computer-implemented method of claim 18, further comprising utilizing the downsafe map and the upsafe map to determine the instructions to be inserted into the routine that set components of the stack.

20. (Canceled)

21. (Currently Amended) For a computer-executable program that operates on a data structure, where the data structure must have a required state at selected program points, a computer-implemented method of transforming said program comprising:

representing a program with a control-flow graph in which actions to take in event of an exceptional situation are represented by exceptional paths in the graph;

analyzing the program to determine at which points a stack must be in a valid state, the stack to represent cleanup instructions for the program;

building a forest of trees that represent a stack state of the stack at said points where:

each node of a tree of the forest of trees represents a possible item on the stack, and

the stack state is represented as a path from the tree node to the root of the tree;

placing operations in the program that set the state of items on the stack based on both of a down-safety calculation and an up-safety calculation for the program;

removing edges from the control-flow graph which represent the actions for the exceptional events; and

inserting ~~a prologue~~ one or more instructions at an entry to the control-flow graph that saves an existing pointer to the top of the stack.

22. (Previously Presented) The computer-implemented method of claim 21, wherein the down-safety calculation is determined by performing:

constructing flow equations for a down-safety of operations that set the state of items on the stack, where the equations ignore branches if it is determined that the branches are fail edges; and

solving the flow equations for the down-safety of the operations that set the state of items on the stack.

23. (Previously Presented) The computer-implemented method of claim 22, wherein the up-safety is determined by performing:

constructing flow equations for an up-safety of operations that set the state of items on the stack, where the equations use the solution for the down-safety calculation to determine which setting operations are speculatively available; and

solving said flow equations for the up-safety of operations that set the state of items on the stack.

24. (Previously Presented) The computer-implemented method of claim 22, wherein the calculation of the down safety further utilizes a lattice that distinguishes strict down-safety from speculative down-safety.

25. (Previously Presented) The computer-implemented method of claim 21, further comprising building a cleanup tree and cleanup states for the program.

26. (Previously Presented) The computer-implemented method of claim 25, wherein the cleanup tree and cleanup states represents the stack at points where an exception may be thrown.

27. (Previously Presented) The computer-implemented method of claim 26, wherein the cleanup tree and cleanup states assist in the placing of the instructions to be inserted into the program that set state of items on the stack.